Functionality ready to land in optimizer and backend

Miguel Garcia
http://lamp.epfl.ch/~magarcia
LAMP, EPFL

2012-05-01

Outline

Status as of M3

- Compilation speed
- `test.stability` under `-optimize`
- Not yet merged: Faster reaching-defs for `ClosureElim`
- Not yet merged: `GenASM`

Ongoing work

- Single-pass Type-flow analysis
- Open performance riddles around inlining

Next steps

Stats compiling the compiler (excerpt, `-Dscala.timings=true`)

phase	id	sec	share(%)
inliner	22	54sec	30%
typer	4	38sec	21%
jvm	26	15sec	8%
erasure	13	14sec	8%
dce	25	12sec	7%
icode	21	8sec	4%
closelim	24	6sec	3%
uncurry	9	5sec	2%
specialize	11	3sec	2%
refchecks	8	3sec	2%
mixin	19	3sec	1%
inlineExceptionHandlers	23	2sec	1%

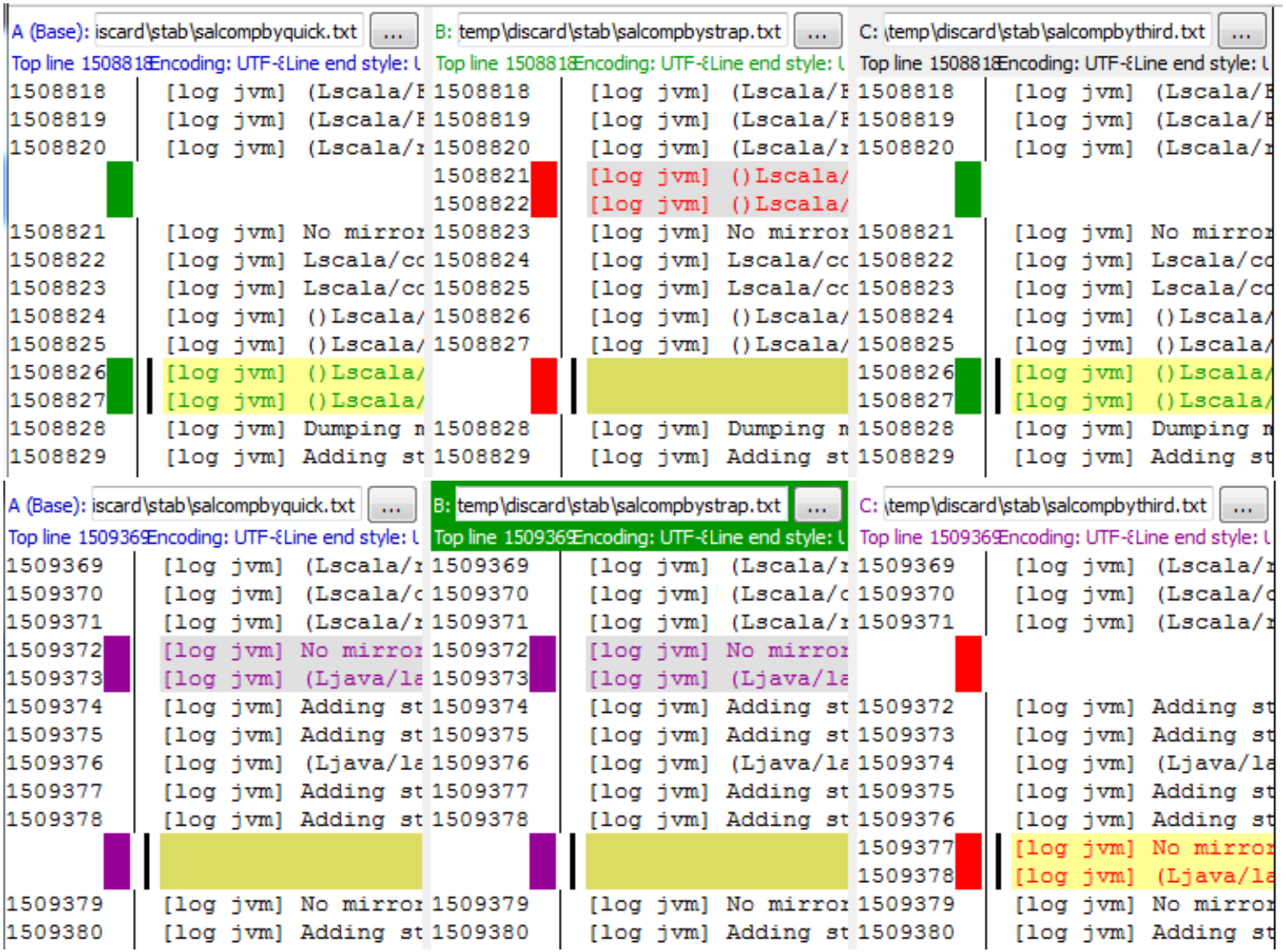
- Observed behavior, with `makePublic()` enabled :
  - `test.stability` fails under `-optimize`.
  - However, for *identical* compiler runs
    (but they really have to be *identical*)
    optimized output is stable (caveat: tested outside `build.xml`).
- Explanation:
  - During `inliner`, there's no topological sorting
    (over the call-graph relationship) of the methods being visited.
  - Different visit orders (across compiler runs)
    lead to different method bodies, with ripple effects.

Recent developments

- `makePublic()` was disabled the night before Scala Days.
- just found that postponing `makePublic()` untill all other inlining conditions succeed, recovers `test.stability`.
- Another idea: sort `IMethod`s by `sym.id`.

Details at: `https://groups.google.com/d/topic/scala-internals/yGm0kBn9Gmk/discussion`

The only non-deterministic behavior that logs show is:



Reaching Definitions

- Status quo: 12sec (but with mismatched stack sizes)
- Fixing that: 25sec
- With some improvements (two `Int`s in a `Long`, etc): 7sec
- Details at `https://github.com/magarciaEPFL/scala/tree/fasterRDef`

47c96c22	magarciaEPFL	no more mismatched stack sizes in reaching def analysis	2 months ago
dcac1d87	magarciaEPFL	lattice meet operation is associative and commutative (at least on te…	2 months ago
32d5d099	magarciaEPFL	towards replacing list scanning with hash operations	2 months ago
ded71a7b	magarciaEPFL	just one iteration over instructions in rdef.init()	2 months ago
72581324	magarciaEPFL	no separate set needed for kill(b), it's always == gen(b).keySet	2 months ago
aac6070f	magarciaEPFL	map replacements doing away with list scans (more to come)	2 months ago
2c7f3d92	magarciaEPFL	using persistent data structures	2 months ago
e55cee9b	magarciaEPFL	dawn of alt reaching defs	2 months ago
57d9320d	magarciaEPFL	one iteration fewer over all instructions rdef.interpret()	2 months ago
628a7ca1	magarciaEPFL	replaced old rdef also in ICodeReader	2 months ago
8a87893e	magarciaEPFL	changes in the management of the worklist	2 months ago
274dd501	magarciaEPFL	stop passing Tuple2[BasicBlock, Int] around	2 months ago

ASM-based backend:

- twice as fast as `GenJVM`,
  even faster with "`Instruction.emit(asm.MethodVisitor)`"
- bootstraps, passes all tests, including:

```
ant quick.clean -Dscalac.args.quickonly="-target:jvm-1.5"
  test.stability
```

  similarly for `jvm-1.5-asm` and `jvm-1.6`

Alternatives for `build.xml`:

1. Download `asm.jar` and `asm-util.jar` from Maven,
   re-namespace on-the-fly viar `JarJar`
2. Distribute ASM sources "re-namespaced" by us

Details at `https://groups.google.com/d/topic/scala-internals/7gecxktTUWs/discussion`

Why bother making inlining faster? Currently:

```
inliner  22  54sec  30%
```

Gist of *Single-pass* Type-Flow Analysis (TFA)

1. `BasicBlock` instructions are scanned at most once, collecting its *net effect* on the output lattice elem ("single-pass")
2. Afterwards, the iterative dataflow uses the net-effects.
3. On average, twice faster as `MethodTFA`.

What can be done with `SinglePassTFA`:

- integrate the *solution repair* approach used in `inliner` (extra 5x speedup)
- more scalable concurrency (lower contention on `typer` as compared to `MethodTFA`)

Morale: avoid repeated pattern matching over ICode instructions.

Details at `https://github.com/magarciaEPFL/scala/tree/SinglePassTFA`

- Riddle 1:

```
def isMonadicMethod(sym: Symbol) = {
  nme.unspecializedName(sym.name) match {
    case nme.foreach |
         nme.filter  | nme.withFilter |
         nme.map     | nme.flatMap      => true
    case _                              => false
  }
```

Any other method "m"
(1) not explicitly marked `@inline` (2) taking a closure as last arg; won't have its callsites inlined thus preventing `ClosureElim` from eliminating anonymous-closure-classes ref'ed at `m` callsites.

- Riddle 2:

```
/* TODO
 * Do we really want to inline inside exception handlers?
 * Seems counterproductive
 * (larger methods less likely to be JITed). */
```

Next steps

- Phasing into trunk the improvements described above.
- Yes, but *which ones* aiming for *which release*? (M4, RC1, 2.11)